Beyond Measure
User Manual and Reference Guide

June 23, 2012

Copyright © 2012 Code Zeal

Table of Contents

Introduction....3
Application overview....4
The Kernel....4
The Configuration Tool....4
The Modules....4
Installation....5
Prerequisites....5
Selecting a database....5
Service Configuration....5
Completing the Installation....6
Using a Trial License....6
Installing a purchased license file....6
Starting the Application....7
General Configuration....8
The Supervisor....8
Logging....8
Advanced debugging....9
Data Link....9
Database Connections....10
Data Sources....10
Modules....11
Module Communication....11
Messaging Overflow....11
Module: 1-Wire....11
Supported 1-Wire Adapters....11
Supported 1-Wire Devices....11
Operation and Configuration....12
Operational Settings....12
Command Priority....13
Setting up adapters and devices....13
Module: Database Storage....15
Module: Graph Generator....16
General requirements and rules for creating a graph....16
Creating Your First Graph....16
Data Series....17
Keywords in Legend Texts....18
Examples....18
Custom Titles....18
Keywords and macros in custom titles....18
Keywords....18
Macros....19
Example:....20
Module: Mail Gateway....21
A Word of Caution....21
Configuration....21
Authentication....21
Port Numbers....21

Copyright © 2012 Code Zeal

Retrieving Mail....21
Configuration Example....22
Google Mail....22
Module: Network Binder....23
Message Distribution....23
A problem: Message Loops....24
Connecting With BM Tool....25
Module: Rule Engine....26
The Basics....26
A First Example....26
Schematics....27
Custom Components....27
Component Library....27
Components....27
Component Properties....27
Input and Outputs....28
Pull States....28
Signal Quality....28
Circular Connections....28
Clock Signals....29
Composite Components....29
Terminals....29
Live View....30
Oscilloscope....31
Tag Components....31
Available Components....31
Module: SMS Gateway....41
Compatible Devices....41
Module: Sound....42
Text-to-speech Setting....42
Installing Additional Voices (TTS engines)....42
Module: Tag Storage....43
The Concept....43
The Module....43
Module: User Module Engine....44
Development Environment....44
User Module Development....44
Life cycle of a User Module....45
Available Message Types....46
Installing a User Module....46
Take care when updating BM....46
Module: xAP Gateway....47
Cron....48
Fields....48
Field Rules....48
Operators....48
Examples....48

Introduction

Contrary to how most other applications work, Beyond Measure (hence forth abbreviated 'BM'),

Copyright © 2012 Code Zeal

does not come prepackaged in a way that it work out-of-the-box. Instead, it lets you, the user, set things up the way you want, and it allows you to purchase only those parts that you want to utilize.

Application overview

BM consists of two main parts:

- The kernel and its modules
- The configuration tool

The Kernel

The kernel runs in the background as a Windows service and is normally not visible to the user. It is responsible for starting the desired modules (modules are parts of the application that provides specific functionality) based on the configuration read from the configuration file.

The Configuration Tool

BM is configured using a tool know simply as “BM Tool” which is the only means to prepare a configuration for use by the kernel. Using this tool, you specify which module(s) to load when the kernel starts, and how they are configured and how they should operate with each other and external systems, if applicable.

The Modules

As stated above, a module is a part of the application that provides certain functionality. Some modules are totally stand-alone, meaning that they have no external dependencies, while others require other modules to be present to provide the full functionality. For example, the DB Storage module (provides a common way for other modules to store data in a database) will perform no work without being asked to do so by another module. A module such as the Rule Engine (which executes a schema containing user defined rules) can be run totally stand-alone without any external dependencies (depending on the rule set, of course). More information on modules is available in the Modules section.

Copyright © 2012 Code Zeal

Installation

Before you can start using BM, you must first install it onto the computer you wish to run it. Start by downloading the latest version from the download page, then start the installer and follow the instructions. Take care to select the upgrade-option if you are performing an upgrade of a licensed system, otherwise the installer will overwrite your license key with a trial key. Before installing however, you should make sure that the prerequisites are met.

Prerequisites

BM runs on any 32 or 64bit platform with Microsoft .NET Framework 4 installed (Windows XP SP3, Windows Vista SP1 or later, Windows 7, Windows Server 2003 SP2, Windows Server 2008 (not Server Core Role), Windows Server 2008 R2 (not Server Core Role)).

Depending on your functional requirements, you will likely have to install one ore more of the following items:

- Maxim's 1-Wire driver for 1-Wire support. (Be sure to select the correct 32 or 64bit version)
- ODBC driver for database connectivity.
    - Microsoft SQL Server
        - SQL Server 2005 (section *Microsoft SQL Server Native Client,* select the x86 or x64 package)
        - SQL Server 2008 (*Microsoft SQL Server 2008 Native Client,* select x86 or x64 package)
    - MySQL
    - SQLite (this driver is included in the application and does not need separate installation.

    Of course, you will also have a functional database server, should you choose other than SQLite.

Selecting a database

As a user new to these things, it might be tempting to opt for the SQLite database as it does not require a separate server installation, but be aware that most users of BM actually uses Microsoft SQL Server or MySQL since they allow for easier access from other applications and also comes with more mature configuration and administration tools.

Service Configuration

During installation the kernel is installed as a Windows service (unless you choose not to) and will run under the local system account. Normally this gives the application sufficient access right to perform its work, but be aware that network access is limited; network shares etc. are not accessible.

To configure the service to run as another user (to allow network share access etc.), follow these steps:

- Open the Services Management Console (Winkey+R, enter "services.msc" and hit Enter.

Copyright © 2012 Code Zeal

- Find the Beyond Measure service in the list.
- Right-click on and select Properties.
- Open the Log On tab
- Select the This account option and enter the account name and password.
- Press Apply. Please note that the changes does not take effect until the the service is restarted (done via the General tab).

Completing the Installation

BM comes with a default, empty, configuration so you may opt to configure the application before starting the service (Winkey+R, enter services.msc, hit Enter then find the service entry and start it) or, you may start the application in desktop mode by using the shortcut on your start-menu. See section Starting the Application for more details.

Using a Trial License

The trial license that comes with the license key allows you to load up to four modules, and a runtime of a certain number of hours since the last reboot of of the computer. Note that each module has its separate time counter, thus each modules will stop operations at different times within the preset time. Once a module considers the trial license to be expired, you have to reboot the computer before operations can continue.

Installing a purchased license file

When you receive a license file after completing a purchase, simply copy it to the installation folder and (re)start the service. The application will look for available license files, and select the one with the latest creation date. Should this search for some reason select the wrong license, such as the trial license (you can determine this by enabling debug log messages) simply delete the license files that you do not want it to use.

Copyright © 2012 Code Zeal

Starting the Application

As noted above, the kernel can run in two modes:

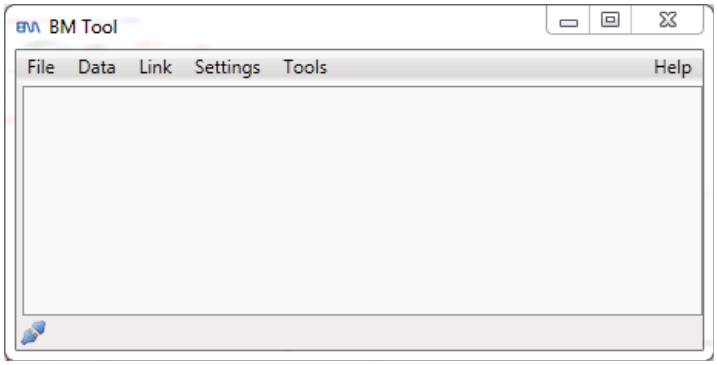
- Desktop Mode
- Service Mode

Running in desktop mode has the advantage that you can see the generated log information directly from within the console window that appears. Regardless of which way you choose, whenever the configuration file is changed, the kernel will reload it and restart all modules so they can refresh their settings. Please note that you cannot run the service and in desktop mode at the same time – only the first instance will be operational.

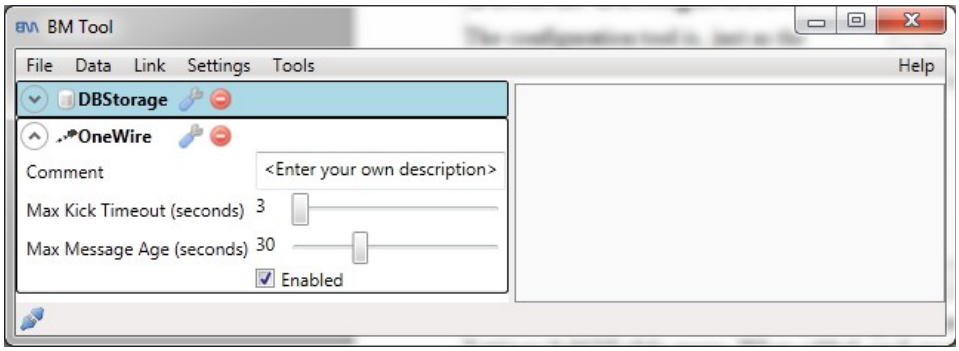
Copyright © 2012 Code Zeal

General Configuration

The configuration tool is, just as the modules, divided into different parts, each with its specific purpose. When first started, it looks like the image to the right. The configuration file read by the kernel/modules is named "BeyondMeasure.scb" and always resides in the installation folder. In all likelihood, you want to open that file before continuing (File/Open menu).



As a new user, you'll want to start by adding the modules you want use, do this through the Settings/Add Module menu. When added, each module instance is displayed on the left side of the window.



Note that you can expand each item to access the settings that modules support. To configure the specific instance, click the tool-icon.

A module can be added multiple times. Reasons for adding multiple instances of the same module type may be that you want to store data in to multiple databases or run two 1-Wire networks in parallel. Some modules makes sense to run in multiple instances while other do not. Care should be taken not to to configure to modules to perform the same work as that usually leads to undesired side effects.

The Supervisor

The supervisor is a part of the kernel that monitors all enabled modules and when they are deemed to be non-functional for whatever reason, will attempt to restart the module in question. Each module reports (aka. 'kicks') to the supervisor with a regular interval, and if it does not it is considered to be non-functional. The most common reason for a module to be considered non-functional is when a module performs a long running operation, such as a database query that results in a large data set. If you know this to happen, you may increase the kick-time for that module. However, the default of three minutes is sufficient in nearly all cases so do not change this without reason. The supervisor also has some configuration options you can change in the Settings/Preferences/Supervisor tab.

Logging

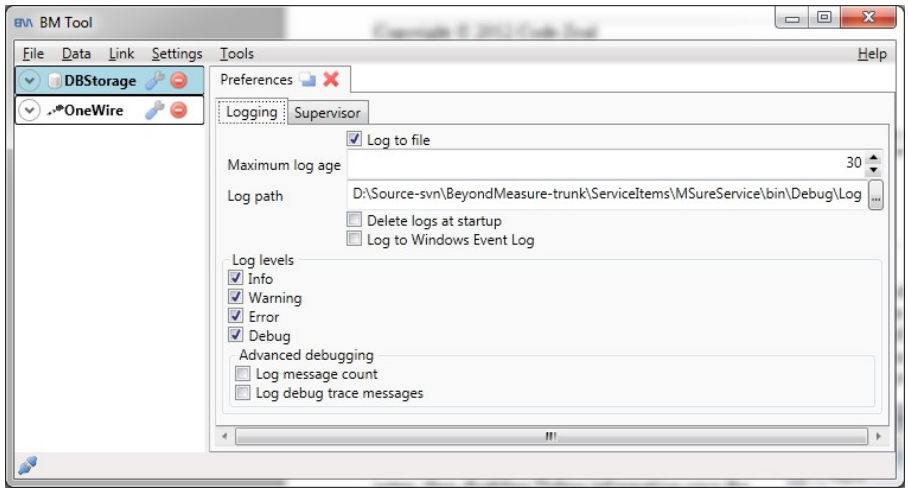
Regardless of run mode (desktop or service), the application will produce log files in the Logs-folder beneath the installation folder. The log files are named after the current date, their respective module name and instance number, or, in the case of the kernel, simply Kernel-<date>.log. These files are UTF-8 encoded files, and as such you are recommended to use an editor capable of reading UF8 encoding (most modern editors do). The application outputs a lot of information to the log files by default (all four log levels are enabled: Info, Warning, Error & Debug) and the log files are the

Copyright © 2012 Code Zeal

first place you should look if you run into problem. It is recommended to let all levels be enabled during initial setup, then disabling Debug information once the system is taken into production use.

You access the Log settings through Settings/Preferences/Logging.

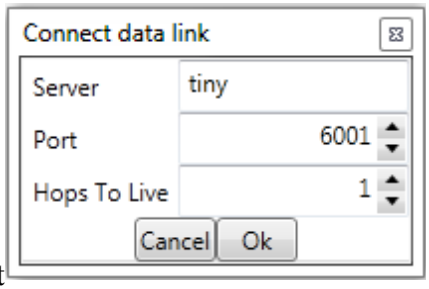
You can select which debug levels you want to log, and whether you want to log them to file and/or Windows Event Log). You can also change how and when log files should be deleted (Windows Event Log clean-up is configured using Windows' own tool).



Advanced debugging

The two options, “Log message count” and “Log debug trace messages” should be used with care as they can potentially cause a huge amount of data being written to the log files and Windows Event log. The latter of them should only be enabled if searching for the cause of a problem in the application.

Data Link



Depending on your use case (such as running a schema with live data), you may want to connect the configuration tool to a kernel running at either the local or a remote computer. To do this you must first enable the Network Binder module in the kernel and configure it to act as a server. Then, open menu Link/Data link/Connect. You will be presented with the Link Settings window in which you must enter the IP address or DNS name of the computer running the kernel. Unless you have changed the port used by the Network Binder module, leave it at its default of 6001. Once satisfied with the server address, press enter and the tool will attempt to connect to the kernel through the Network Binder module. The data link indicator will reflect the link status in the lower left corner of the window. For information about the Hops To Live-setting, see Module: Network Binder.

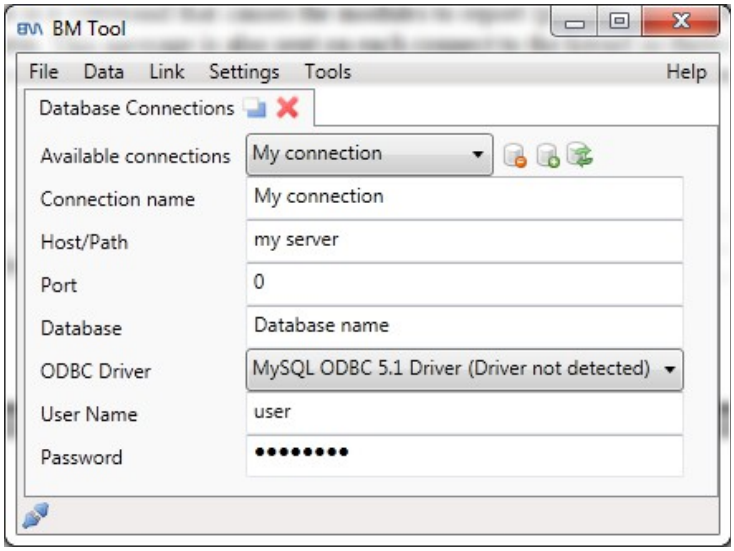
There are two additional menu items under the Data Link menu; “Block all outgoing non-user commands” and “Request Update”. The former can be used to prevent a schema running in the configuration tool to interfere with a live system by preventing any command messages being sent to the kernel. The latter one is a command that causes the modules to report (possibly re-read from end devices) its current status. This message is also sent on each connect to the kernel so there is no need to manually issue the command without good reason as it may cause an unwanted load on the system.

Copyright © 2012 Code Zeal

Database Connections

Connections to databases are configured centrally using the Database Connections window (Data/Database Connections).

Each connection requires a set of parameters:



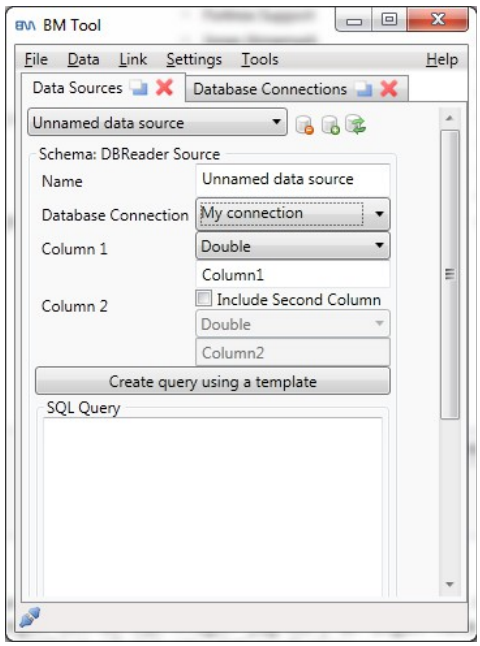
- Connection name
- Host/Path (in the case of SQLite databases, you must enter the path to where the file should reside on the local computer)
- Port (not used for SQLite)
- ODBC Driver (if the driver cannot be located, it is indicated in the drop down box)
- User name
- Password
- Database name (in the case of SQLite databases, you must enter the name of the database file (so the full path to the database becomes the path entered in the Host-filed + the database name))

You can click the icon beside drop down box to test the current connection. The application will attempt to create the database if it does not already exist. Of course, the creation of the database will only work if the specified user account has sufficient access rights in the database.

These connections can later be used from various modules.

Data Sources

Data Sources (menu Data/Data Sources) are used to define how to retrieve data from external sources, such as a database, for use in for graph generation. Each data source requires a database connection from where the data should be read and information on how the data should be parsed. There are a number of templates for different 1-Wire sensors, but you may also write your own custom SQL queries.



Currently, there are two types of data sources:

- Graph Sources – these are for use in Data Series in graph generation.
- Schema Sources – these are for use in schema components.

Copyright © 2012 Code Zeal

Modules

A module (also known as a “plug-in” or “add-on”) can be seen as a separate application, but one that runs within the same application context as other modules. This separation into function-specific parts makes it easier to build a system that can be expanded based on the user's need without adding parts that are not used from the start.

Module Communication

All modules are capable of talking to each other through a common messaging interface; this enables great possibilities in terms of integration. To the average user these messages are of no immediate concern, but for those that delve into the User Module Engine, these messages are of great importance. The kernel handles all message distribution automatically so there is no need to manually configure it; as long as the module is loaded, it can receive and send messages to/from other modules.

Messaging Overflow

All messages takes a certain amount of time to process. For example, a command to read a 1-Wire device takes approximately one second to execute, depending on the network complexity. If you request it to be read every half second the receiving module, in this case the 1-Wire module, will start accumulating a number of unprocessed commands, essentially delaying the execution of the commands. To prevent the system from being flooded, all modules drop messages that are too old (by default 30 seconds or older) and logs a warning in the module log.

If any of your modules logs such a message, it means that it cannot keep up with the number of incoming messages and you need to revise your system configuration to lighten the load.

For each separate module, you can set a maximum message age. Although you can set it as high as a full minute, doing so is not recommended unless you know that a module occasionally is busy for such a long time.

Module: 1-Wire

The 1-Wire module adds the ability to read and control Maxim's 1-Wire devices. For those that are not yet familiar with these, Maxim provides a technical but good introduction here. In less technical terms, a 1-Wire device can be seen as a component with a very specific functionality that is controlled by a master, in this case the 1-Wire module of BM.

Supported 1-Wire Adapters

BM supports those adapters that are supported by Maxim's 1-Wire drivers, except for parallel port adapters.

- Serial: DS9097U
- DIY/Homebrew adapters (various schematics can be found on the Internet)
- USB: DS9490R

Supported 1-Wire Devices

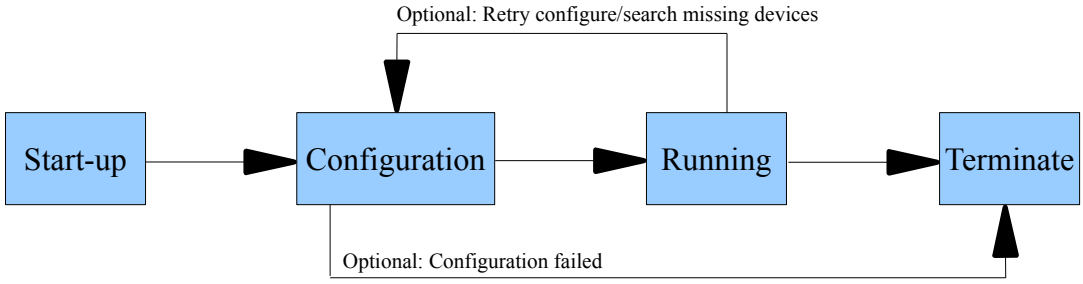
- 0x05 – DS2405

Copyright © 2012 Code Zeal

- 0x10 – DS18S20
- 0x12 – DS2406 (one and two channel versions), DS2407
- 0x1D – DS2423
- 0x1F – DS2409 (when acting as a 1-Wire network switch)
- 0x20 – DS2450
- 0x22 – DS1822
- 0x26 – DS2438 (also as humidity devices when paired with humidity sensors HIH4000/4021)
- 0x28 – DS18B20
- 0x29 – DS2408
- 0x30 – DS2760/61/62
- 0x3A –DS2413
- LCD devices: There are currently two supported LCD devices; Swart and Hobby-board. Since neither of these devices have an official family code, they must be manually converted from their actual device types from within the configuration, and their type and LCD-size provided by the user.

Operation and Configuration

There are four states in which the module operates, and they are linked as shown in the following figure.



During the Start-up state, the 1-Wire network is scanned for present devices, including devices behind 1-wire network hubs. This search has been verified to a depth of two cascaded hubs, but should work at deeper depths too. When the search is complete, the module enters the Configuration state in which it configures all devices according to the settings defined by the user. Such settings are device specific and includes settings like alarm levels and value resolution. When all devices have been configured, the Running state is entered in which the devices are read according to the defined schedule and commands from external sources are processed.

Operational Settings

1-Wire networks are mostly stable, but it does happen that a device temporarily is unavailable for various reasons, especially on larger networks with long wires. There are two options to handle such situations.

- *Automatically search for missing devices every 15 minutes.*

Copyright © 2012 Code Zeal

- With this option enabled, the module will enter a search-mode every 15 minutes when a device is unavailable during start-up.

- *Instead of terminating when device configuration fails, retry every 15 minutes.*
  - With this option enabled, the module will resume normal operation even when the configuration stage fails for a device during start-up.

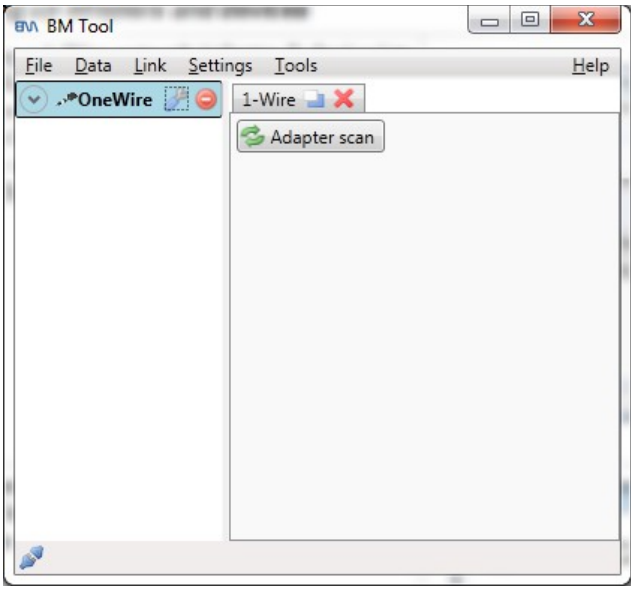
Command Priority

Commands that are received from external sources, such as a component in a schematic, have priority over the normal read schedule and as long as there are commands to process, the normal read schedule is put on hold. In practical terms, this means that if too many commands are sent to the module it will not be able to perform its normal procedure. A command does however not abort a scheduled read already on progress when the command is received.

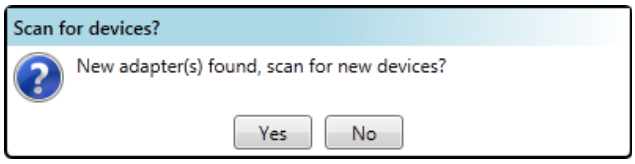
Setting up adapters and devices

Adding a 1-Wire network (adapter & devices) to a configuration is done through the use of the auto-search function. Simply press the button labeled “Adapter scan”. When you do, the application will search all available ports (Serial and USB) for available adapters.

If a serial adapter is found, the application will ask if it is of a DIY/Homebrew type in order to handle it correctly

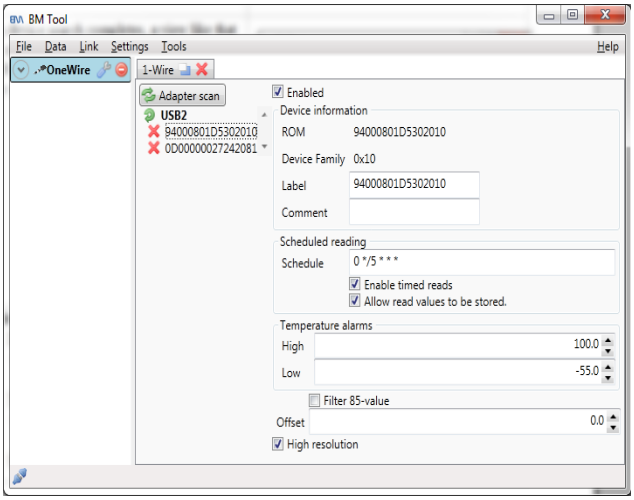


When the adapter search is complete, a question asking if a search for adapters should be performed is shown. Select Yes to search for all connected devices on the found adapters.



Copyright © 2012 Code Zeal

When device search completes, a view like that to the right is shown with all devices listed on the left side and the properties for the selected device on the right side.



Each device type has its own set of properties, and there are a few common properties for all devices:

- Enabled – if unchecked, the device will not be considered for any operation.
- Device information
  - Label – specifies the name to use throughout the application when identifying the device. Also used as database table name.
  - Comment – allows for storing an arbitrary text along with the device, such as its location.
- Scheduled reading
  - Schedule – specifies the read interval for the device. This setting uses the Cron format.
  - Enabled timed reads – if unchecked, the device will never be read according to the schedule, only on direct request from an external source.
  - Allow read values to be stored – if checked, a properly configured DB Storage module will store the read values from this device.

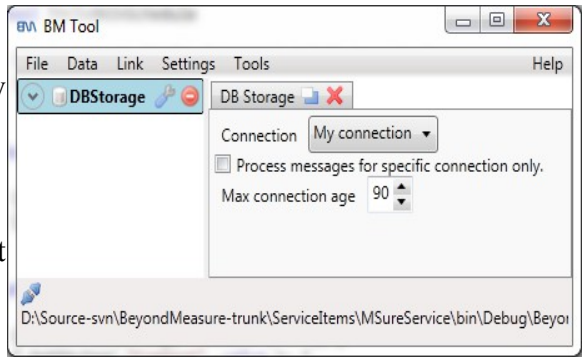
Copyright © 2012 Code Zeal

Module: Database Storage

This module has only one purpose; to store data gathered by other modules in a database. It requires only a configured database connection to be operational. Once activated, the module will translate messages sent from other modules into tables and columns in the database. Only one database connection can be selected, but you may add additional instances of the module to store the data into multiple databases.

It can operation in two modes:

- Processing all general storage requests from other modules
- Processing only those requests that specify a specific database connection.



The latter mode is used together with a DB Writer component configured with a specific database connection. The reason for the second mode is to allow for configurations where the DB Reader component decides where a value should be written and not the Database Storage module.

The second mode has one limitation that it is important to be aware of; it does not function between different instances of BM linked together with a Network Binder module, even if the database connections are configured with the same name.

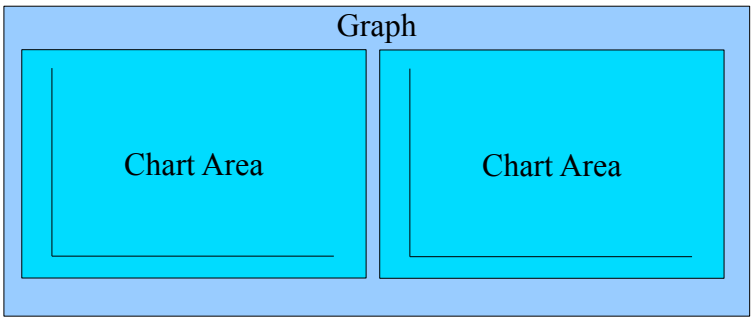
Regardless of operational mode, all connections used by the module are kept alive for the configured time. By setting this to a time slightly longer than the shortest interval between two write requests it is possible to keep the number of reconnects to the database server to a minimum, thereby improving performance.

Copyright © 2012 Code Zeal

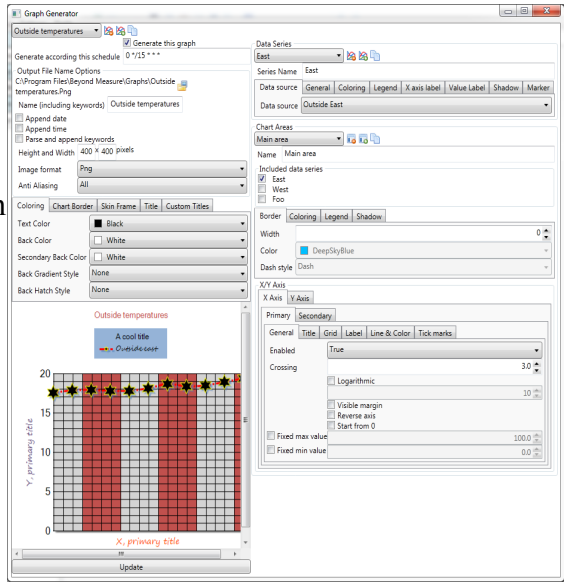
Module: Graph Generator

This module is for generating static graphs according to a user defined schedule. Almost any data series can be plotted; as long as the data is retrievable as described in Data Sources it should work.

Each graph image is structured as illustrated below. The graph in turn is contained within the actual file that is written to disc. The user interface for the graph module follows the same structure as the graph image – remember this and you will have an easier time finding the settings you need.



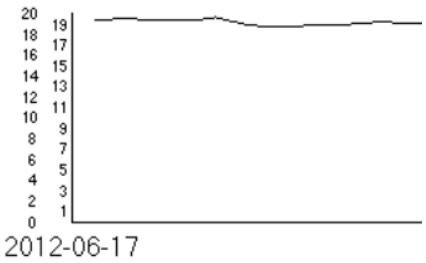
General requirements and rules for creating a graph



- Data Series are what is plotted in the respective Chart Areas. A Data Series can be reused multiple times within the same graph.
- A Chart Area requires at least one Data Series.
- A Data Series requires a Data Source, see section Data Sources, from which it will retrieve the data that should be plotted.

Creating Your First Graph

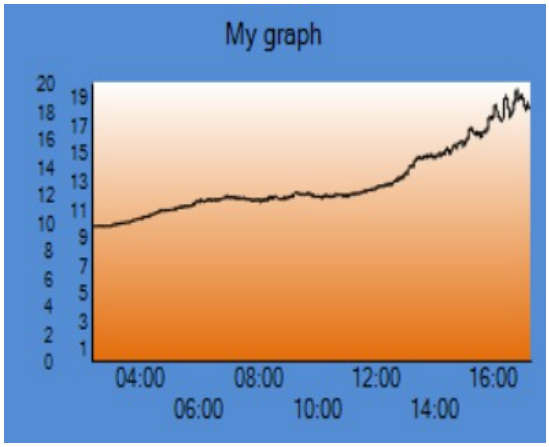
1. First, make sure that you have a functional Data Source, for example one that retrieves time stamp and temperatures from a table in a database.
2. Add a new graph by clicking on the icon with the +-symbol.
3. Give the graph a name, this also specifies the name of the output file. Note that you can use 'keywords' to insert the current time etc (see the tool tip for a list).
4. Under “Data Series”, select the data source to read values from. You probably want to change the default name of the data series.
5. Under “Chart Areas”, check the data series to include.
6. Now press the button “Update” in the lower left corner (you might have to scroll down). A very basic graph is now shown, much like the image to the right.



Lets make the graph a bit more eye-pleasing.

Copyright © 2012 Code Zeal

8. Select the following file option:
    1. Anti Aliasing: All
9. Coloring
    1. Back Color: Cornflower Blue (R:100, G:149, B:247, A:255)
10. Title
    1. Title: Enter a title for your graph.
11. Chart Areas
    1. Coloring: A gradient of your liking and then two different colors for “Color” and “Secondary Color”
12. Chart Areas / X/Y Axis
    1. Under tab “Label” enter “HH:mm” (without quotation marks) into the format string box.
    2. Enter “1” in the Interval box and select Hour in the Interval Type box.
13. Press the Update button in the graph render window and you should have something like this image, depending on what your data and color options were.



Hint: You can duplicate graphs, this can for example be used to try different settings without changing the original graph configuration.

Data Series

Data Series (abbreviated 'DS' below) are what is plotted in a chart area. They have many options, most of which are described by their name alone. However, there are some things that need further explanation.

Chart Type	Specifies how the DS is plotted; as a line, area etc. If you have multiple DS in the same Chart Area, not all combinations are possible. For example, bar and columns cannot be used in the same Chart Area and when using Stacked types each series must have the same number of data points.
X-Axis/Indexed	When checked, the data points are indexed along the X-axis instead of being placed by its actual value. If enabled, and two or more series are enabled, both must have the same number of data points.
Gradient	If a Gradient is enabled, it takes precedence of any other color for the same area. Cannot be used together with Hatching.

Copyright © 2012 Code Zeal

Hatch	The hatch color is taken from the color for the same area. Cannot be used together with Gradient.

Keywords in Legend Texts

Each data series has the property “Legend Text” where you may enter a text that will be displayed in the legend for each chart area. By using certain keywords, you can extract values and calculate certain mathematical expressions. The allowed keywords are:

#TOTAL	Total of all Y values in the series.
#AVG	Average of all Y values in the series.
#MIN	Minimum data point of all Y values in the series.
#MAX	Maximum data point of all Y values in the series.
#FIRST	First data point of all Y values in the series.
#LAST	Last data point of all Y values in the series.

All keywords allows the usage of format strings. Format strings are appended to the keyword and always begins and ends with an opening and closing brace: #KEYWORD{#.##}

There are several format strings available, you can read the full specification here and here, but for most use cases a combination of the #, 0 and .-character will be sufficient.

Examples

#AVG{#.##}	Round the average value to two decimals
#LAST{00.00}	Display the value using at least two digits
#FIRST{#}	Round the value, no decimals

Look here for more examples.

Custom Titles

Under tab Chart Appearance/Custom Titles, you can add your own title boxes and place them where you want on the graph.

Keywords and macros in custom titles

The custom titles supports several keywords and macros that results in auto-replacement with actual values.

Keywords

{DATE}	Returns the current local date.

{TIME}	Returns the current local time.
{UTCDATE}	Returns the current UTC date
{UTCTIME}	Returns the current UTC time.
{yy}	Year, two digits
{yyyy}	Year, four digits
{MM}	Month, two digits
{dd}	Day, two digits
{hh}	Hour, 12h format
{HH}	Hour, 24h format
{mm}	Minute, two digits
{ss}	Seconds, two digits
{PATH}	Returns the path, excluding the file name to where the graph is stored.
{FULLPATH}	Returns the path, including the file name to where the graph is stored.
{NAME}	Returns the name of the graph, as specified in the graph configuration.
{HEIGHT}	Returns the configured height of the graph.
{WIDTH}	Returns the configured width of the graph.

Macros

%MEAN('source')	Returns the mean value of the Y-values for the given data source.
%MEDIAN('source')	Returns the median value of the Y-values for the given data source.
%LASTVALUE('source')	Returns the last value of the Y-values for the given data source.
%MAX('source')	Returns the maximum value of the Y-values for the given data source.
%MIN('source')	Returns the minimum value of the Y-values for the given data source.

Copyright © 2012 Code Zeal

Example:

%MEAN('outside, west')

results in the mean value of the data source which has its Name-element set to “outside, west”

Note:

Values returned by these macros are always retrieved from the Y-axis of the data source.

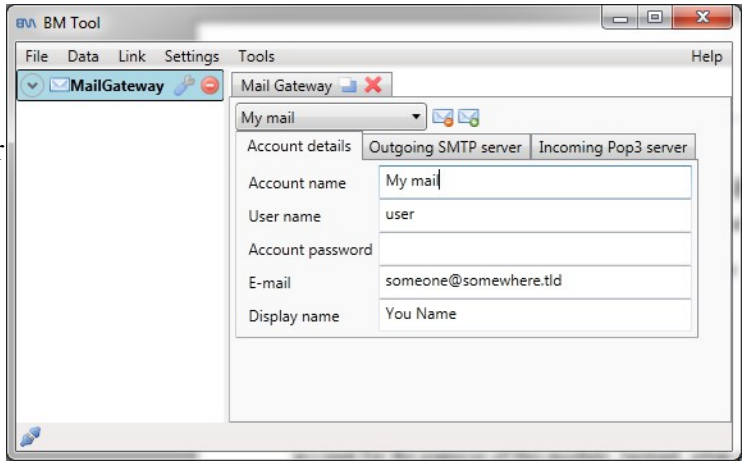
The single quotation marks must be included in the configuration, thus is %MACRO('source') valid syntax, %MACRO(source) is invalid.

Hint: By inserting “\n”, you can force a line break in the text.

Copyright © 2012 Code Zeal

Module: Mail Gateway

Please note: To utilize this module, you also need the Rule Engine or the User Module Engine.



E-mail is great thing and with today's smart phones and tablets it can be used in many ways, such as for notification and/or controlling units connected to your system. The purpose of this module is to act as a gateway to one or more e-mail accounts, allowing e-mail to be both sent and received from within BM.

A Word of Caution

We recommended against using your regular e-mail account for the purpose of this module. Instead, setup a separate account dedicated for use with this module.

Configuration

Authentication

All serious e-mail providers requires the e-mail client to authenticate before being allowed to send e-mails in order to prevent their systems from being used by anonymous spammers. This module supports two modes of authentication:

- Normal authentication using user name and password.
- Pre-authentication using POP3. When enabled, the module will first log in to the POP3 server using the provided account details before attempting to send an e-mail using SMTP.

Although the module allows both to be be activated at the same time, one should be sufficient. Ask your ISP which one that applies to you.

Port Numbers

The standard SMTP port number is 25, but most ISP/e-mail providers blocks this port and tend to to configure their SMTP servers to use another port, often 2525, instead.

The standard port for POP3 is 110. When using a secure POP3 connection (by checking the appropriate check box) the port should usually be changed to 995, but this may vary so check with your ISP/e-mail provider.

Retrieving Mail

As each mail is received from the server, it is transformed into a format that can be distributed internally between modules. Note that attachments are not supported at this time.

You may configure each separate account to for manual (instead triggered though a rule schema) or automatic, periodic, retrieval of e-mails in 1 minute intervals. Entering a 0 will disable the automatic retrieval. To filter which messages that shall be retrieved and parsed, you may enter a

filter string that must be present anywhere in the subject of the e-mail (case insensitive comparison).

Note: If you choose not to delete the email after retrieval, the same messages will be read over and over until you manually delete them.

Configuration Example

Google Mail

Google provides all the required information to configure BM to work with GMail on this page but in short, the following settings have been seen to work:

Outgoing:

- SMTP Server: smtp.gmail.com
- Port: 587
- Pre-authenticate using POP3: No
- Login using credentials: Yes
- Enable SSL: Yes

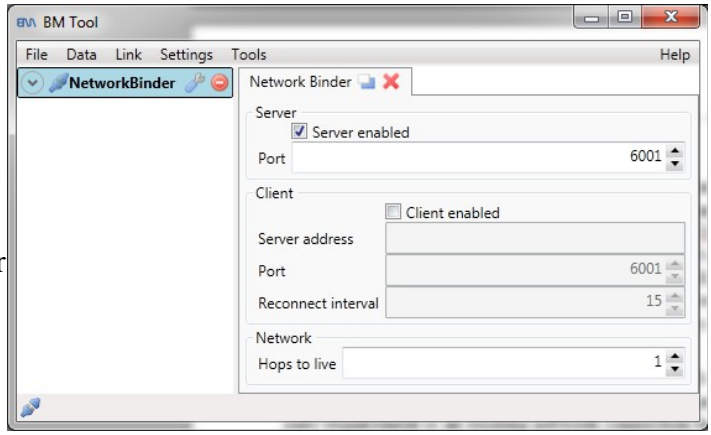
Incoming:

- POP3 Server: pop.gmail.com
- Port: 995
- Enable SSL: Yes

Copyright © 2012 Code Zeal

Module: Network Binder

Sometimes, it is desired to connect two or more instances of BM to build a larger system out of smaller parts. For example, lets say you have a low-power computer (connected via WiFi) running in your garage for monitoring of temperatures and light control and a media server in your main building running a second instance for additional monitoring and control. Connecting the two instances would enable you to use devices from either instance in a schema as if they were part of the same system.



This is where the Network Binder comes into play. It enables you to connect a number of instances together into a larger system, the only requirement is an existing network connection between the physical locations; be it WiFi, hardwired, GPRS/3G or other media.

It provides to operational parts:

- Server - other instances of the Network Binder can connect to it.
- Client - the Network Binder connects to a remote server instance.

Hint: This module provides a great benefit when designing schematics in Sanity to be run in the Rule Engine module, as Sanity can connect and receive live data from the server giving you live feedback directly in the editor.

Message Distribution

All messages are distributed to all connected clients/servers, as well as internally in the local BM instance with two restrictions:

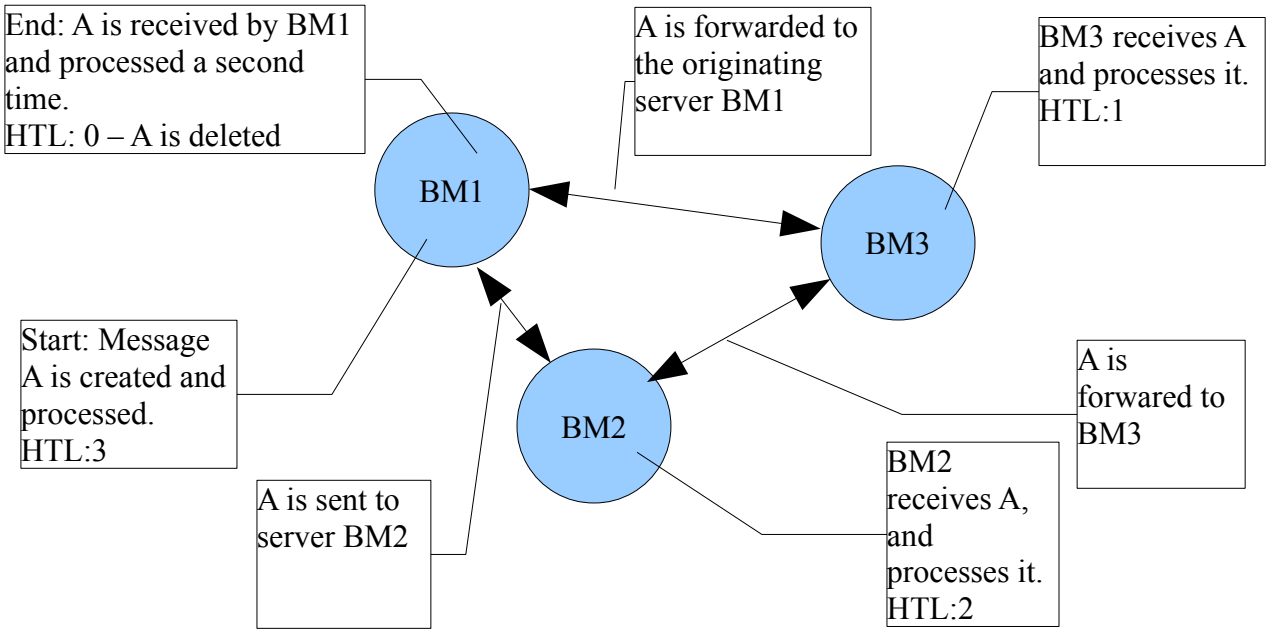
- Hops To Live (HTL). A 'Hop' is the transference of a message between client/server A and B (illustrated as an arrow below). Each time a message hops from one to the other, the messages' HTL-counter in decreased by one and when it reaches zero, the message will not be distributed further.
- A message is never sent back the way it came to prevent rouge messages in the system.

Copyright © 2012 Code Zeal

A problem: Message Loops

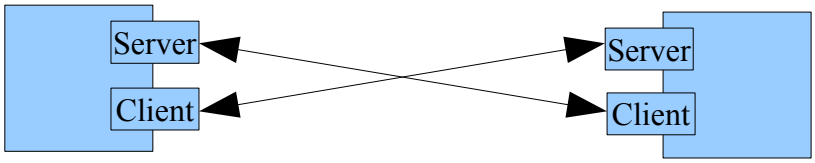
Despite the above two distribution restrictions, you can still end up with messages doing loops within your system if you configure your Network Binder instances to form a circle connection, as illustrated below. The problem is in that the server BM1 processes message A a second time.

This image does not show the full problem though, as the message A is actually distributed from BM1 to both BM2 and BM3, which means that before the message is deleted, it is processed twice


End: A is received by BM1 and processed a second time.
HTL: 0 – A is deleted
A is forwarded to the originating server BM1
BM3 receives A and processes it.
HTL:1
BM1
BM3
Start: Message A is created and processed.
HTL:3
BM2
A is forwared to BM3
BM2 receives A, and processes it.
HTL:2
A is sent to server BM2

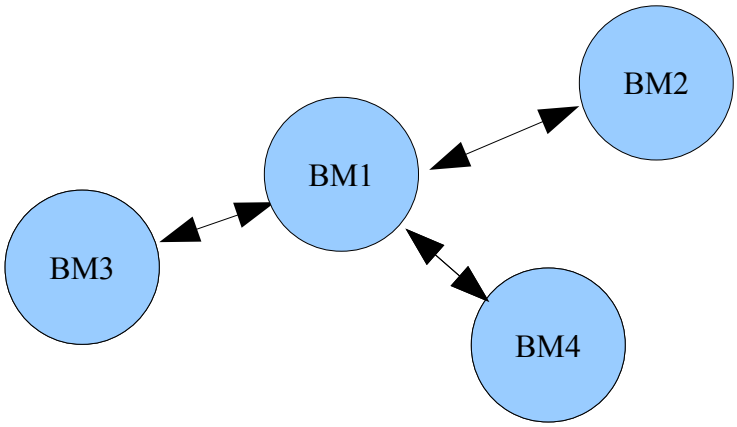
by BM2 and BM3, and as many as three times by BM1 before the message is deleted from all systems.

Even with only two instances, it is possible to get this problem if you cross-connect two instances of this module:

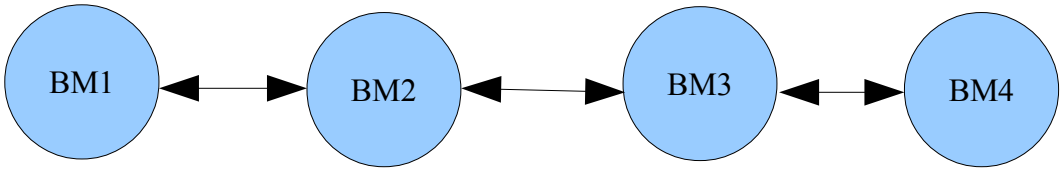


To prevent this situation from happening, the default setting for HTL is 1, thus eliminating the problem in most installations as a message is never forwarded when received from another system. Should you need to connect more than two systems, you can either connect them in a star network, or in series. Depending on which one you choose, the HTL-setting must be updated for each system.

Copyright © 2012 Code Zeal



Star network. BM2, 3 & 4 requires HTL: 2, BM requires HTL:1



Series network. BM1 & 4 requires HTL: 3, BM2 & 3 requires HTL:2

Connecting With BM Tool

To which Network Binder instance should BM Tool connect, and what HTL setting should be used in the above two scenarios?

Remember that the transfer to the tool adds one Hop. This gives the following answers:

- Star network: Connect to BM1 and set HTL in Sanity to 2.
- Series network: Connect to either BM2 or BM3. Set HTL in Sanity to 3.

Copyright © 2012 Code Zeal

Module: Rule Engine

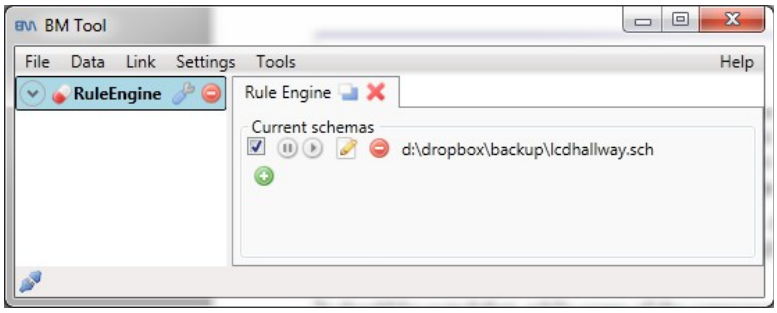
In 1854, George Boole developed what is known as Boolean algebra. Today you take advantage of his work almost without noticing it – any electronic device with a microprocessor (such as a personal computer, smart-phone or calculator) uses Boolean algebra to perform its tasks. Now you actively can take advantage of his work through the use of the Rule Engine module.

This module really embodies what BM stands for - flexibility and the user's own choice. It brings together different worlds, such as 1-Wire, xAP and SMS into a single domain where the user can make them interact in a way he or she wants.

It should be noted that, while some of the components included with this module functions without external dependencies, many requires other modules to be activated and configured.

The Basics

This module do not perform any operations on its own, instead it loads a set of user-built schematics that define the operations to be performed based on input from internal or external sources. Of course, it is entirely possible to write a schematic that is fully self-sustained if the requirements are such. Adding and removing schematics to load is done via the module configuration, as for any other module.

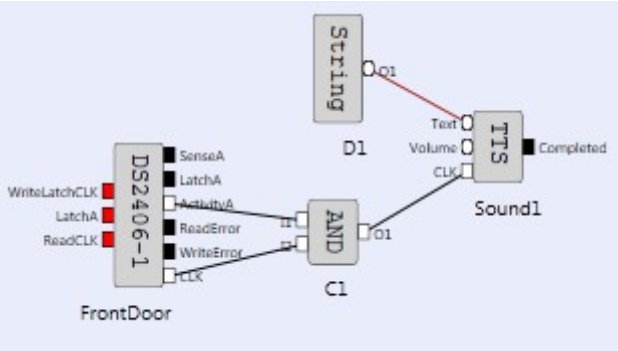


The Play and Pause buttons sends a message to a running instance of the Rule Engine module (via Network Binder) to unload or load the specified schematics. This only works if it is known to the module in the currently running configuration. The Edit button naturally opens the schematics in an editor.

A First Example

Lets say that we have a 1-Wire network switch (DS2406) connected to a push button on the front door, and when someone presses that button we want to have our computer say the phrase “There is someone at the front door”.

A schematic that realizes this functionality can look what is illustrated in this image.



In short, it functions as follows:

1. The 1-Wire module reads the status from the device labeled 'FrontDoor'.
2. The Rule Engine receives the resulting status message and forwards it to the FrontDoor

Copyright © 2012 Code Zeal

component.

3. If activity was detected on the input of the switch, the output ActivityA will go high and a clock pulse emitted on the CLK output. Together, they the are AND:ed (*C1*) into a clock pulse for the TTS component, *Sound1*, which will send a command to the Sound module to output the phase read from the component's *Text* input; the String component (*D1*) holds the phrase to speak.

Schematics

As seen above, a schematic is a set of components connected via wires to perform a specific function. A schematic is in itself a special type of component – a Composite Component, see chapter Composite Components, but labeled as a 'schematic' since it is the topmost/encompassing component. You use the schematics editor (accessed via menu *Tools/Rule Editor*) to create your schematics.

Custom Components

By saving a Composite Component to the folder <*installation dir*>*ComponentLibrary*, you can make your own components available for reuse from the library. If you want to organize your own components into separate sections in the library, simply create sub folders under the *ComponentLibrary* folder, and they will be shown parallel to the *Custom*-section.

Note: When a custom component has been added or removed from a the library, you must reload the library through the menu *Library/Reload* from the editor's menu.

Component Library

The component library, on the left side of the editor, is where all available components are held. These are added using drag-and-drop into an editor. The components are sorted by their type, such as Logic (AND, OR), Terminals, module-specific etc and can be mixed as you see fit in a schematics.

Components

A component can be seen as a box that, given some input, performs a specific operation and outputs the result on its output pin(s) and/or as a message to an external part, such as another module. Some components are self-sustained, i.e. they are not dependent on external parts, while other components requires another component or module to perform some operation before performing its own operation. To rename a component, just double-click on the component name and enter the new name.

Most components have a clock-input, and often a clock output, pin. These pins are used to trigger an operation, or to signal that new data is available on the outputs of the component as a result of the physical device being read.

Component Properties

When open, the *Component Properties*, on the right side of the editor, shows the properties for the currently selected component. All components have a comment property (including the schematic itself). Many components have additional properties that can be adjusted to fit the requirements.

Copyright © 2012 Code Zeal

Input and Outputs

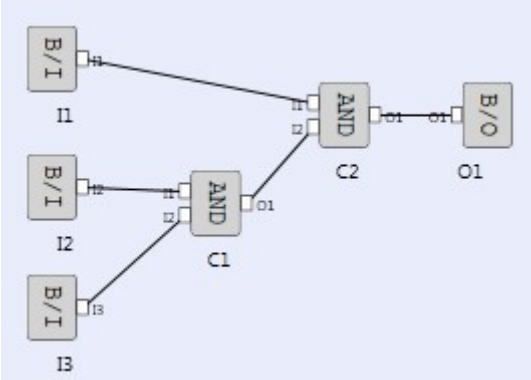
Most components have both inputs and outputs. Inputs are always located on the left side of the component and outputs on the right side. In- and outputs (commonly known as connection points – see Terminals) are used to pass signals between two components, via a wire. There are two kinds of connection points:

1. Boolean – these are used to signal either true (high) or false (low) signals. These connection points are visualized as a square.
2. Data – these are used to pass arbitrary data, such as a string or a temperature value. These connection points are visualized as a circle.

Pull States

All *unconnected* Boolean inputs have a default pull-state (default value) of *false*, as indicated with red coloring. By *pressing and holding shift and right-clicking* on an input, the pull-state is toggled between *true* (colored green) and *false*. When a wire is connected, the pull-state does not have any effect.

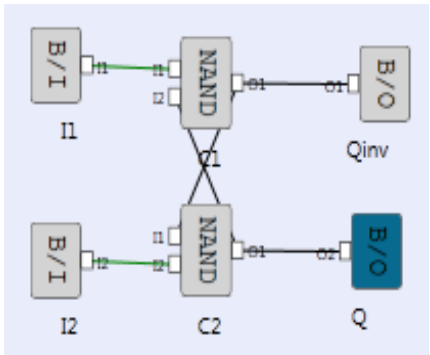
Signal Quality



Depending on how a schematic is built, there are times when the state of the input/outputs are unknown. This is known as a signal having *bad quality*. For example, look at the schematic in the image (a user-built 3-input AND gate). This schematic has three inputs, *I1, I2* and *I3* and one output, *O1*. Since neither of the inputs has a connected wire (ignoring the default pull-state), they have neither a *true* nor a *false* value, thus the quality is bad. Bad quality is visualized with a black coloring of the connected wires.

When a component has an input with a bad signal quality, the component will in most cases not operate until all inputs has a signal with good quality.

Circular Connections



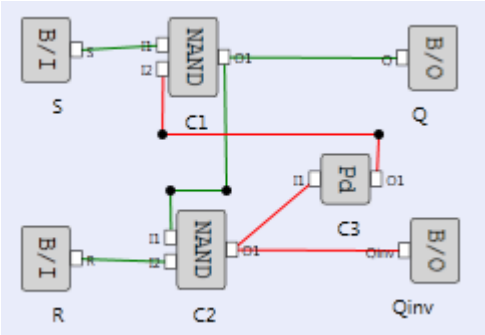
When designing a schematic, one should take care not to create circular connections. The image illustrates a non-functional $\overline{\text{SR}}$-FlipFlop circuit. The reason it is not functional is due to the circular connections between components *C1* and *C2*. As each component is dependent on the others output, neither is able to provide a good signal quality on its output. To solve this issue, a *pull-down* or *pull-up* component can be used as seen in the next image.

The *Pd*-component pulls its output low when it receives a bad quality on its input pin, thus making it possible to connect as desired. As can be seen in the image, the wires are now colored red and green, depending on the logical value.

It should be noted that circular connections have a negative impact on performance due to that they cause a recursive evaluation of the state of the components. If such a situation is encountered, the evaluation will be forcefully stopped after a certain number of recursions to prevent infinite loops and subsequent application malfunction.

Copyright © 2012 Code Zeal

Clock Signals

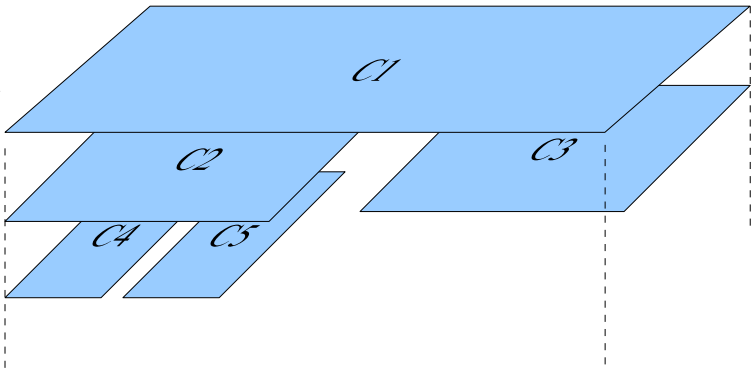


Many components have a *CLK*-input, where CLK stands for "clock". These components do not perform their operations based on changes on their other inputs, if any. Instead, they require a clock-pulse (low to high state change) to be provided on their clock input. When such is detected, the component perform its operation.

Some components also has a *CLK*-output. When such is the case, they first update the values on their other outputs, if any, then they provide a clock-pulse on their clock output. By using the clock pulses, it is easier to determine when something will happen in a chain of components.

Hint: Components without a clock input perform their operation directly whenever their input changes state or value.

Composite Components



A composite component is a component like any other component, with the difference that it can contain any number of other components in different layers. For example, the component C1 in the image contains two components, C2 and C3. In turn, C2 consists of C4 and C5. In this example, C1 and C2 must be composite components (they contain other components) but C3, C4 and C5 can be either a composite component or a regular component.

Hint: You can dive into a composite component within the editor by *ALT-clicking* on a component. To go back up a level, *ALT-click* on an empty space in the editor.

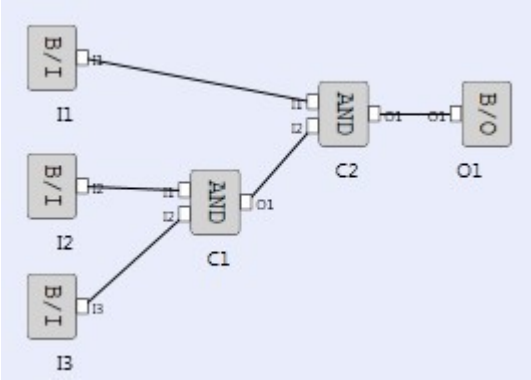
Terminals

In order to build a reusable component, and to connect different layers inside a composite component, some sort of connection points are needed where wires can be attached. This is where terminals come into play. Just like connection points, there are two types – Boolean and Data terminals, of which there are both inputs and outputs.

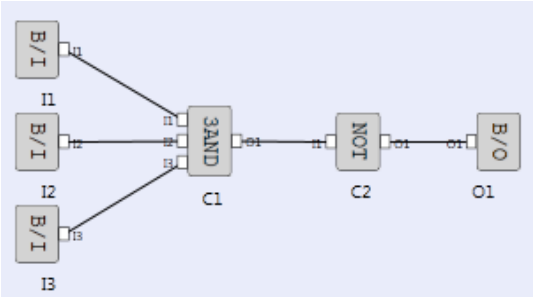
You've already seen the terminals in use in the Input and Outputs section, but we will now look at them in more detail by building a simple three-input NAND component.

Copyright © 2012 Code Zeal

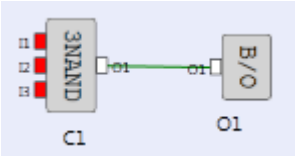
First, we build the three-input AND component. The Boolean inputs (B/I) will serve as the input connections when using the schematic as a component. Likewise, the Boolean output (B/O) will serve as the output connection.



After saving the component in the Error: Reference source not found (named 3AND), add it to a new schematic and connect three input terminals and a NOT-component and also a final output terminal as in the image.



Again, save the schematic as a new component in the Error: Reference source not found (named 3NAND), and add a copy of it to a new schematic. You have now built a composite component in two layers. An additional output terminal has been added in the image, just to illustrate that the output of the 3NAND component in fact has a value of *true*.



Hint: By right-clicking on an input terminal, you can set a temporary value it should provide on its output. This way, you can test a schematic directly within the editor.

Live View

When BM Tool is connected to the kernel using the Data Link/Network Binder, your schematic is run directly from within the editor as if it was running under the Rule Engine module. This way, you can edit and test the schematic and directly see the results based on actual data.

There are a few important items you need to be aware of:

1. The schematic is run as the current user so access rights etc. may be different than when run under the Rule Engine.
2. If BM Tool is not run on the same computer as the Rule Engine any differences between the two may affect the outcome.
3. If the schematic is active in the Rule Engine at the same time as it is run from within an editor, there is a risk that they may interfere with each other. For example, double sets of commands may be sent to an external system – one from each instance. To avoid this, either unload the schematic in the Rule Engine module, or prevent outgoing messages from leaving Sanity through the *Link/Data Link/Block all outgoing non-user commands* menu.
4. There is no synchronization made between a running instance of a schematics under the

Copyright © 2012 Code Zeal

Rule Engine and the same schematics opened in an editor – they are completely separate instances.

Oscilloscope

The oscilloscope (accessible via *Tools/Oscilloscope*) can be a great aid in viewing and troubleshooting a schematic. You add (*right-click*) a probe to either input or output connections and whenever the value on that point changes, the oscilloscope will display the new value. A probe is removed by *pressing and holding CTRL and right-clicking* on the input or output.

Boolean data is always presented as either 0 or 1, while a data point will be presented as its actual value, but only if it is convertible to a numeric value. Values that are not convertible can be viewed in the 'Data' tab, where all values are presented in text format, with the most recent value at the top.

It may appear the the oscilloscope do not update, but that is due to the fact that it does not unless a new value is detected; this behavior allows for easier readings. A history of maximum two minutes between the first and most current value will be viewable, values older than that are removed.

Tag Components

A Tag is a user-defined pair of a name and a certain value. Using a tag, you can abstract a physical device into a logical one. For example, if you use a 1-Wire or xAP device to monitor the state of your front door, you can distribute the resulting value as a Tag, thereby removing the need to know how the value is read by the system.

Tags can also be used to create your own named "signals" for communication inside a single, or between multiple schematics.

Note that tags of different types do not interact with each other. So you can have multiple tags named "A" as long as they are of different types. Having multiple *output* tags of the same type with the same name is strongly discouraged as it usually creates unwanted side effects. Having multiple *inputs* of the same type and same name is however not an issue.

Available Components

Category	Component name	Description/Notes
1-Wire		These components represents a physical device on a 1-Wire network, and requires its physical device to exist to function. Each component is named by its official name. To pair the schematics component with the physical device, rename the component in the schematic to match the label or ROM code of the device in your configuration. All 1-Wire components have a pin named ReadCLK used to request a read of the device. One example usage is to control the read schedule via a schema instead of from the 1-Wire module's schedule configuration. All 1-Wire switches have a WriteLatchCLK input to trigger a write-command to the latches. DS1822 DS18B20 DS18S20

	DS2405	
	DS2406-1	Single channel version of DS2406
	DS2406-2	Dual channel version of DS2406
	DS2408	
	DS2413	
	DS2423	
	DS2438	
	DS2438 Humidity	When you configureyour DS2438 as a humidity device, you can use this component to use the humidity values in a schematic.
	DS2450	
	DS2760/61/62	
	LCDHobbyboards	
	LCDSwart	Due to a limitation in the firmware of the device, it does not support reading latch states on the outputs and thus does not support the synchronization procedure used with with other switch-devices.
Data	Compare	Compares the input values according to the specified operation. If both values can be converted to a number then numerical comparison is used, otherwise culture invariant string comparison used used.
	Counter	Increases/decreases a counter in the direction and step specified for each clock pulse. Emits a clock pulse after each change.
	Data select	Selects between two input values based on the Select-value. Low, selects value at input D1.
	Math	Performs mathematical operations.
	Shift	Shift register. Data is taken from input I1, to output O1, then O2 on each clock pulse. Enable determines if the component is active. Reset clears the outputs. Can be chained to create shift registers holding more values.
	String	Holds a single string as input to other components.
	Strlen	Outputs the number of characters of the input string.
	Text Format	Formats up to four strings/values into a single string at its output. • The component has four inputs, D0 through D3. These are known as argument 0, 1, 2 and 3 in the format string. • The input “TryNumeric” is used to tell the component if it should try to treat the input values as numerals instead of plain strings. • The Pad-inputs allows padding the result up to a fixed length (PadLeft/Right) using a specified character (PadChar)
	Transform	Transforms the input value according to the specified transformation and decimal properties.

	Value	Hold a single numerical value as input to other components.

	XML Parser	Used to retrieve a value from a specific path/tag and/or attribute within a well-formed XML data block.
		
		• Inputs
		  ◦ XML- XML data to read
		  ◦ CLK – triggers the parsing
		• Outputs
		  ◦ Value – contains the value retrieved from the found element. In those cases where the path results in an an element with sub elements, the full XML structure from the current element is made available for further processing in an other XML Parser.
		  ◦ Attribute - contains the value retrieved from attribute from the found element.
		  ◦ CLK – pulses when a new block of data has been parsed.
		• Properties
		  ◦ Path – the path (including the root element) to the desired element.
		  ◦ Instance count – used to select one of the many instances of the desired element (sorted by document order)
		  ◦ Attribute – the name of the attribute on the found element to retrieve.

Sound	Clear Sound	Clears the current and all enqueued sounds for the module which the Target Module Instance Name property matches.
	Sound File	Plays the specified file on the module which the Target Module Instance Name property matches. If provided, the file provided on the input takes precedence over the one specified in the properties.
	TTS	Speaks the specified text on the module which the Target Module Instance Name property matches. If provided, the file provided on the input takes precedence over the one specified in the properties.
Time	Date Compare	Provided with start and end dates, in the form of month and day, its *Within*-pin indicates whether or not the current date is withing the provided two dates. The *CLK* output pulses on every change of the *Within*-pin. The *UTC* input pin tells the component to use local or UTC times. If the provided month and day specifies an invalid date, such as 29:th of February 2011 (2011 is not a leap year), the component will silently reduce the day and month in 1-day decrements until the date is valid. E.g. 29:th of February 2011 becomes the 28:th 2011. If the end date is less than the start date (e.g. end = 1/1 and start = 1/12) then the end date is considered to be during the following year.

Copyright © 2012 Code Zeal

	Delay	Delays the input signal with a configurable delay, including a configurable random element. Whenever a positive flank on the input clock is detected, a time interval is calculated and a countdown is started. When the countdown reaches zero, the signal on the input is read and output on the output signal pin and a clock pulse is generated on the output clock. If the input signal is changed during a countdown, the countdown interval is recalculated and a new countdown is started. The time interval is the product of the number of intervals and the interval length +/- a percentage of the calculated interval, as specified in the configuration of the component. For example, a 50% randomness and an absolute time interval of 1 minute results in actual intervals ranging from 30 seconds to 1 minute and 30 seconds.
	Sun	Signals when the sun is up or down based on provided longitude, latitude, UTC offset and Zenith.
	Time	Provides either UTC or local time on its outputs.
	Time Compare	Provided start and end times, its *Within*-pin indicates whether or not the current time is within the provided time range. The *CLK* output pulses on every change of the *Within*-pin. The *UTC* input pin tells the component to use local or UTC times. If the end time is less than the start time (e.g. end = 01:00:00 and start = 22:00:00) then the end time is considered to be on the following day. If a valid time is provided on the input pin *Time*, this time is used as the base for calculations, otherwise the component defaults to the system time.
	Time Delay	Adds or subtracts a random time interval from the date/time provided on the *BaseTime* input, whenever a clock signal is received in the input clock. The resulting date/time is provided on the *Time* output.
Database	DBReader	Using a special type of data source (see Data Sources), this component reads the first row in the resulting data set and outputs the values onto the O1 and O2 pins (if enabled in the source), and then pulses the CLK out pin. If the operation succeeds, the *Result* pin will be high during the clock pulse, otherwise low. Note that due to the possibly memory- and time-consuming nature of the database read operation, once a read request has been triggered by a positive flank on the CLK input, no further read operations will be initiated until the first one has completed.

Copyright © 2012 Code Zeal

	DBWriter	This component is used to write a value to a database table. It requires a Database Storage module, with a matching configuration. There are two operational modes: • By selecting the “Specific database connection“, you can specify a specific database connection to use when the value should be stored. • By not selecting a database connection, any DB Storage module (not configured for the above mode) that receives the write request will process the message. The expected columns in the table are: • index (64 bit integer, auto incrementing) • Timestamp (DateTime) • One column as specified in the component properties. If the database and/or table doesn’t already exist, it will be created by the DB storage module (assuming sufficient access rights). The Result pin will indicate the result of the write request: • As reported by the DB Storage module when an attempt to write the provided data has been made • As a result of the preparation of the request (value conversions etc.), in which case only failures can be reflected. (success can only be reported by the DB Module, see (1)) The out clock is pulsed every time the Result pin can be read. Please note that, depending on your configuration (such as multiple active DB Storage modules) you may get multiple pulses with different result indications for each write request.
File I/O	File Monitor	Can monitor a file, or files, including sub directories, for changes and output the path and file name when a change is detected.

Copyright © 2012 Code Zeal

	File Reader	Using regular expressions it can read a file one line at a time and output up to four groups of data extracted from that line. • The component keeps a track file on disk that allows it to continue where it left off last. • As the reader keeps track of the last read position by using a tracker file paired with the file to be read, you should not use multiple readers for the same file as doing so will cause the readers to access the same tracker file and thus overwriting each others read positions. • When it goes high, the *CLK* input triggers a new read cycle of the file specified with the *Path* and *Filename* inputs. • If high during the clock pulse, the Reset input will reset the read position to the beginning of the file. If this pin is held constantly high, the file will be read from the beginning on each clock pulse. • The Require new line character" property specifies if the line must end in a new line before being parsed. Example Using the regular expression, without quotes, "(?<g1>.{1,})\:(?<g2>.{1,})\:(?<g3>.{1,})\:(?<g4>.{1,})\;" would match the line, without quotes, "a:b:c:d;" If g1, g2, g3, g4 had been entered in the component properties for Group 1-4, the values "a", "b", "c" and "d" would have been made available on the outputs.
	TextOut	Can write data to a file.
Logic	AND	AND operation.
	CLK	Provides a clock pulse with the approximate interval specified by the properties.
	D.CLK	Delayed clock. This component will generate a clock signal after the configured interval whenever a clock pulse is sensed in its input. If another clock pulse is sensed before the interval expires, the interval starts over again without triggering a clock pulse on the output.

Copyright © 2012 Code Zeal

	D.Out	Transfers the input value to to the output value with a delay specified in the properties. Up to 50 changes are queued, after that the oldest change event is removed.
	NAND	Not AND operation.
	NOR	Not OR operation.
	NOT	NOT operation.
	OR	OR operation.
	Pass through	Used for testing schematics. Normally just passes the input value to the output, but may be locked to high/low output using the context menu.
	Pd	Pull-down. This component is the opposite to the Pull-up.
	Pu	Pull-up. This component give a *true* signal on its output whenever the signal quality is bad on the input. Otherwise it will just give the same signal as that on the input.
	Shift	Shift register. Value is taken from input I1, to output O1, then O2 on each clock pulse. Enable determines if the component is active. Reset clears the outputs. Can be chained to create shift registers holding more values.
	T.P	Triggered Pulse. This component gives a pulse (*false-true-false)* whenever the input signal changes.
	XOR	Exclusive OR operation.
Mail Gateway	Mail	Together with the Module: Mail Gateway, the Mail component adds the ability to send and receive e-mail from any number of different accounts. You need to set the account name to use in the component properties, otherwise the Mail Gateway won't be able to match the send-request to an e-mail account. You can send the same e-mail to multiple recipients by separating the input string on the To-pin with ';', e.g "one@domain.tld;two@domain.tld". Each send attempt will result in a clock signal on the *ResultCLK* pin, with the result indicated on the Result pin. If the Mail Gateway is configured for manual retrieval, use the *RequestCLK* to initiate a retrieval of messages from the configured account, otherwise set this pin to a pull-down state (indicated by red)
SMS Gateway	Modem	In conjunction with Module: SMS Gateway the Modem component provides the means to send and receive SMS messages and to act on incoming call notifications.

System	Execute	This component can start an an arbitrary process/application (.exe, .bat etc.) • Inputs ◦ The input clock triggers the execution when going from low to high. • Outputs ◦ ExecuteCLK – pulses when the configured command is successfully executed. ◦ ErrCLK – pulses when execution fails for whatever reason. ◦ ExitValue – if *Wait for exit* is selected, the exit value of the process will be available on this port when *ExitCLK* pulses. ◦ ExitCLK – pulses when the process has exited and *Wait for exit* is selected. Note that, when *Wait for exit* is selected, a new process will not be started until the previous one has exited. If Wait for exit is not selected, the component will start a new process on each clock pulse, which can lead to multiple running processes if care is not taken.
Tag	BT/I	Boolean Tag Input – acts as an input for tags from the current or another schematic. The property Tag Name (case insensitive) specifies the name of the tag the component should listen for and should match the name specified in a tag output component. “Validity time” determines how old a value may be before it is considered invalid. When passed, the default value is provided on the output, and the Expired output pulses once. Any change on the outputs is followed by a clock pulse. See Tag Storage for information about the concept of Tags.
	BT/O	Boolean Tag Output The counterpart for the Boolean Tag Input.
	ST/I	String Tag Input Same as Boolean Tag Input, but for strings.
	ST/O	String Tag Output The counterpart for the String Tag Input
Terminal	B/I	Boolean Input. Creates an input pin in a composite component.
	B/O	Boolean Output. Creates an output pin in a composite component.
	D/I	Data Input. Creates an input pin in a composite component.
	D/O	Data Output. Creates an output pin in a composite component.

Copyright © 2012 Code Zeal

xAP

BSC stands for Basic Status and Control. Devices with the word "mirror" in the name mirrors the state of a physical device (such as a Netiom) and can also control it. The rest of the devices lives inside the schematic, but are also visible on the xAP network. Read more about xAP here.

BSC Level Input Mirror

BSC Level Output

BSC Level Output Mirror

BSC State Input Mirror

BSC State Output

BSC State Output Mirror

XAP Reader

This device is capable of reading any xAP schema and retrieving one value for each output pin from a block/key pair. For example, take the following xAP message.

xap-header
{
v=12
hop=1
uid=FF400100
class=Weather.Report
source=mi4.weather.1
}
Weather.Report
{
tempf=19.2
utc=21:59:52
tempc=-7.1
icon=Dry
date=20110214
}

By configuring the component with the addresses and uid, it is possible to retrieve the values from the key/value pairs within the Weather.Report block simply by entering the block and keys into the component properties.

Copyright © 2012 Code Zeal

	XAP Writer	This component is the opposite of the reader. By specifying block names and keys, an xAp message is built with the data retrieved from the inputs. You can specify the same block multiple times to add more than key to the same block.
Web	Web Request	Based on the configuration, this component reads a web page and uses the configured expression for a match in the retrieved value. An example usage is to read a string from a web page for further action in the schema, or to report a value to a web server using the standard http protocol.

- Inputs
  - I1 through I5 can be used to replace the tokens '$1′ through '$5′ withing the URL property with data from outside the component, such as values read from external sources.
  - CLK – triggers the retrieval of the configured URL. Only one retrieval can be active at any one time, so a high clock frequency does not result in multiple requests.
- Outputs
  - Value – If the request succeeds, the data retrieved from the URL, after the matching against the value expression, will be available on this pin.
  - ValueFound – if a match is found then this pin will be high, otherwise low.
  - Result – reflects the result of the retrieval of the URL; high for success, false for failure.
  - D1 – contains the raw data retrieved from the URL or if the retrieval fails, a message that may provide some information on what when wrong.
  - CLK – pulses when data on the outputs have been updated and is ready for reading.

Copyright © 2012 Code Zeal

Module: SMS Gateway

With a compatible cellphone or modem, the SMS Gateway brings the world of SMS, caller id and caller mode (voice/fax) into the schematics and User Modules.

It allows you to send and receive SMS messages, detect incoming calls, either with or without caller id and perform actions based on these events.

Adding/removing devices is done though the context menu (right-click) in the Devices list on the left hand side. Although the default settings works with most devices, there are some differences that might require you to make some adjustments, especially regarding the message settings. You will however always need to enter the COM-port number to which the device is connected. It can be found in the properties for the device via Windows' control panel.

You can set a maximum number of SMS messages to be sent for each device you add. When this limit has been reached, no more messages will be sent during the current 24h-period or until the module is restarted.

Compatible Devices

Any cellphone or modem that follows the GSM07.05 standard (ESTI, RE/SMG-040705PR5) should be compatible, but some may still not be due to vendor-specific differences. The device must also support communication via a serial port (either directly or via a Bluetooth or similar adapter).

The following devices have been known to work

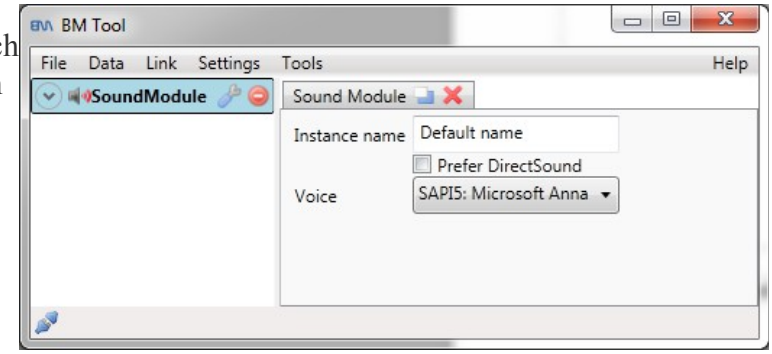
- Mobile Phones
  - Sony Ericsson
    - K700i
    - P1i
- Modems
  - Multitech
    - MTCBA-G-F1
      - Requires the use of the "SR" storage setting.
  - Cinterion
    - MC55i

Copyright © 2012 Code Zeal

Module: Sound

This module adds the ability to play sound files and text-to-speech functionality.

Each instance of this module takes an instance name in its configuration, which is used to determine if a command from one of the sound components should be processed or not. If you leave the instance name empty, or set to "*" (no quotes), the module will process all commands, regardless of their actual target instance (see the descriptions for the components for more details).



When a command is processed, and it is determined that the sound should be played, the module will use the default sound device, as configured in Windows. The volume is determined by the system's overall volume setting, in addition to the volume specified in the command.

The module can only play one sound at a time, and all received commands (up to 30 in total) are held in a queue until all previous sounds have been played, or a command to clear the play queue is received.

Text-to-speech Setting

The module allows you to select among the compatible voices installed on your system. If no voice is selected, no TTS operation can be performed. There are two supported TTS-interfaces; COM (Component Object Model) and SAPI5, where the latter is preferred. The drop down box will indicate the type of interface the voice is accessed through. Some voices are accessible through both interfaces, but may be named differently.

Installing Additional Voices (TTS engines)

Any engine you install must be either SAPI5-compliant, or support the COM (Component Object Model) API. It is recommended to evaluate the compatibility with BM using a trial before buying the full product.

Two providers that provide downloads for SAPI5-compliant trial versions are: IVONA and Cepstral.

Acapela Group laims to provide SAPI5-compliant voices for personal use, but their voices are not listed as such by the .NET framework. However, by using COM-API, these voices are still usable in BM.

Copyright © 2012 Code Zeal

Module: Tag Storage

The Concept

The idea behind Tags are two-fold: to provide a way to separate physical devices from logical functions and to provide a way to create your own 'signals' that can be shared between different levels of a schematic, and even between separate schematics.

Why would you want to separate the physical device from the logical function? Imagine that you have a device of an arbitrary type which monitors the state of your front door (open/closed). Wherever you want to use this status you are not interested in *how* it is read (*physically*) but only what the actual reported value is (the *function*). So instead of using the status directly from the physical device (e.g. xAP or 1-Wire) when you create a Tag named 'front door' and read that tag wherever needed. You have now gained two benefits: 1) replacing the physical device will be much simpler (since it exists in just one instance) and 2) you can set it to a default to a value used before the physical device has had a chance to report the actual value (during start-up) and a maximum age before reverting to the default value, should the physical device ever stop reporting its default value.

Creating your own tags can greatly simplify your schematics since they are shared both within and between schematics.

A Tag does not have an expiry time. Instead, they know when they where last updated/created and it is up to the reader of the Tag to determine how old a value may be before using the default value instead.

Tags sent between multiple computers

The last-update time stamp of a tag is set by the creator of the Tag. The time used is always that of the local computer. Normally you do not have to consider the implications of this, but if you transfer tags between to or more computers using the Network Binder module, it is important that both computers have synchronized their clocks, otherwise the time calculations for the tags will give unexpected results. A typical example of when this situation may occurs is when running a schema within a Rule Editor in BM Tool when connected to a kernel on another computer.

It is recommended to set up all affected computers to synch their clock against an NTP-server to avoid any problem related to unsynchronized clocks.

The Module

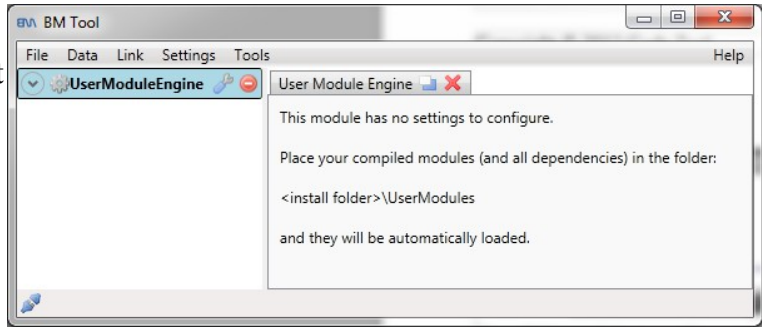
The purpose of this module is to provide persistent storage for Tags between different sessions (computer restarts etc.) and to oversee distribution of existing tags whenever a third-party (module or otherwise) requests an update. This module has no configuration options and works entirely in the background once enabled.

Copyright © 2012 Code Zeal

Module: User Module Engine

Use of this module generally requires knowledge of programming in C#.

A User Module (hence forth abbreviated UM) is a piece of compiled source code that runs within BM and that can interact with other modules through the use of the common messaging system and also leverage the power of the entire .NET framework. The User Module Engine is responsible for handling the loading and communication with the configured UMs.



Development Environment

A UM is written in C# under Visual Studio (The free Express or a commercial version) or another development tool of your own choosing. This manual assumes that you are using Visual Studio.

User Module Development

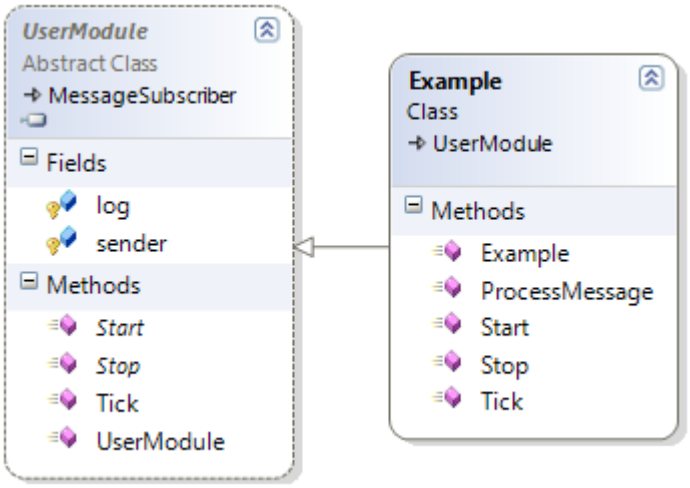
During installation, a Visual Studio Solution is installed in the <install-dir>\UMDevelopment folder, containing an example User Module and the required items to run it from within Visual Studio for debugging purposes as if they were run inside the Rule Engine Module, with the exception that they are run on your development machine (if different from the computer running BM) and that they are run as the user account used by Visual Studio. Note: A running instance of the Network Binder (in server mode) is required on the machine you want to connect the development environment to.

The default solution consists of the following parts:

UMDevelopment.sln	The solution file
UMDevelopment.csproj	The development tool/UM runner making it possible to run your own UM outside of BM.
ExampleModule.csproj	Example User Module project. When you wish to create your own project from scratch, you must reference the following assemblies: • BMMessage.dll • UserModuleBase.dll • PluggableLogger.dll Output type must be Class Library. You must also add a reference to the new project in the UMDevelopment project so it can find the new classes.
Program.cs	The main application that sets up your environment during development.

Copyright © 2012 Code Zeal

UMEnviroment.cs	Provides the API/Enviroment required by all User Modules. You should never edit this file.
Example.cs	An example User Module.



- Messages from other modules are handled in the different overloads of the *ProcessMessage()* method.
- All UM runs within the same thread, so lengthy operations should be avoided whenever possible. Use a separate thread or other asynchronous method to handle such operations. Not honoring this requirement may cause a performance penalty and increased memory footprint, and eventually too old messages being dropped by the UME.
- All UM must inherit from the UserModule class, and be decorated with the UM attribute, otherwise they are not discoverable by the UME or development environment.

```
[UM]
public class Example : UserModule
{
	public Example()
	{
	}
}
```

You will likely want to change the first code line in the Main() method of the Program-class of the development project so that it connects to your Network Binder module (assuming you want to communicate with other modules).

```
// Create the User Module environment, configured to connect to a
// Network Binder module running at the local host on port 6001
UMEnviroment ev = new UMEnviroment( "localhost", 6001 );
```

During development, the environment will automatically load and instance all UM found in the 'Modules'-folder located in the same folder as the project and solution files. The the example project has its output folder set to that directory already.

Life cycle of a User Module

1. Discovered and instanced by the UME or development environment.

Copyright © 2012 Code Zeal

2. Start() is called.
3. Approximately every 100ms, a call to Tick() is made.
4. Incoming messages are passed to the UM through the ProcessMessage() overloads.
5. Repeat 3 - 4 until the application is terminated.
6. Stop() is called.

Available Message Types

All message types that can be sent or received lives within the BMMessage assembly, namespace BeyondMeasure.Message and other namespaces beneath it. It should be noted that most of these message types should not be sent from a UM as that can cause unwanted side effects so use care.

Installing a User Module

When you want to run the User Module from within the User Module Engine, follow these steps:

- Compile the UM in either release or debug mode.
- Place the resulting assembly (.dll file) in the 'UserModules' folder beneath the installation folder.

When next (re)started, the User Module Engine loads and starts the added UM.

Take care when updating BM

Whenever you update BM, you should make sure that your user module are compatible with the new version in terms of the messaging API. You should at least recompile all UM against the new assemblies installed by the update.

Copyright © 2012 Code Zeal

Module: xAP Gateway

xAP is an open protocol intended to support the integration of telemetry and control devices primarily within the home (quote). In BM you can create virtual xAP devices and also create mirroring devices that represent xAP devices that exists outside BM. This module is only a gateway for components created within a schema so to harvest the power of xAP you must also enable and configure the Rule Engine.

All devices on an xAP network requires a unique identifier and address (details), and the xAP Gateway is no exception. You will need to enter a UID that matches either the v1.2 (NNDDDDSS) or v1.3 (NN.DDDDDDDD:SSSS) format.

The default UID, FF0100, may have to be changed not to conflict with an existing xAP device on your network. If you change it, remember to set the sub-uid part to all zeros. The default instance name is the same as the network name of the local computer.

Copyright © 2012 Code Zeal

Cron

Cron is a very flexible format used to specify a series of times when an event is to take place. This format is mostly known from the Crontab service found in Unix/Linux (the equivalent of Windows' Scheduler). BM uses an extended version of this format, adding the possibility to specify seconds in addition to the other five time parameters.

Fields

Each cron-setting has the following fields

Second Minute Hour Day-Of-Month Month Day-Of-Week

where

- Second is 0-59
- Minute is 0-59
- Hour is 0-23
- Day-Of-Month is 0-31
- Month is 1-12 (January = 1, December = 12)
- Day-Of-Week is 1-7 (Monday = 1, Sunday = 7)

Field Rules

- Each field is separated by a space-character
- It is not allowed to insert a space-character inside a field value; “1 -2”, “1,2, 3” and so on are invalid.
- All fields, except Day-Of-Week are mandatory. If present, Day-Of-Week takes precedence over Day-Of-Month, unless specified as '*'.

Operators

There are four operators available:

- Comma (',') operator is used to specify a list of values, for example “5,7,9”.
- Dash ('-') operator is used to specify a range of values, for example “2-8”.
- Asterisk ('*') operator is used to match all valid values. For example, an asterisk in the minute field means “every minute”.
- Slash ('/') operator (known as the “step”-operator) can be used together with the asterisk operator to skip a given number of values. For example, “*/2” in the hour field would mean “every second hour”, or “all hours evenly dividable by two”. “*/2” is equivalent to “0,2,4,6,8,10,12,14,16,18,20,22…58". Note that 0 is evenly dividable by any number.

Examples

Fields	**Description**
* * * * *	Matches every second
0,30 * * * *	Matches seconds 0 and 30 every minute, same as */30 * * * * *

Copyright © 2012 Code Zeal

*/30 * * * *	Matches seconds 0 and 30 every minute, same as 0,30 * * * * *
* 15 * * *	Matches every second every 15:th minute past the hour.
0 */15 * * *	Matches four times every hour; second 0, minutes 0, 15, 30, 45
* * * 1-5 *	Matches every second of every hour during the 1st—5th of every month
15 * * 1-5 7,8	Matches the 15th second of every minute during the 1st—5th of months July and August
10 40 13 * *	Matches 13:40:10 every day
0 0 0 * * 1-2	Matches midnight (00:00:00) of every Monday and Tuesday
* * * * * *	Equivalent to * * * * *

Copyright © 2012 Code Zeal